Conditional Learning of Rules and Plans by Knowledge Exchange in Logical Agents

Stefania Costantini[1], **Pierangelo Dell'Acqua**[2], and **Luís Moniz Pereira**[3]

[1] Dip. di Informatica, Università di L'Aquila, Coppito 67010, L'Aquila, Italy
`stefania.costantini@univaq.it`

[2] Dept. of Science and Technology - ITN, Linköping University, Norrköping, Sweden
`pierangelo.dellacqua@liu.se`

[3] Dept. de Informática, Centro de Inteligência Artificial (CENTRIA), Universidade Nova de Lisboa, 2829-516 Caparica, Portugal
`lmp@di.fct.unl.pt`

Abstract. This paper is related to logical agents and in particular discusses issues related to learning sets of rules from other agents. In principle, the approach extends to agent societies a feature which is proper of human societies, i.e., the cultural transmission of abilities. However, the new knowledge cannot be blindly accepted and incorporated, but should instead be evaluated (and thus possibly discarded) according to its usefulness. We propose a technique and its formalization.

1 Introduction

Adaptive autonomous agents are capable of adapting their behavior according to changes in the environment, by means of some kind of approach. As it is widely acknowledged, the effects of learning should include at least one of the following: (i) the range of behaviors is expanded: the agent can do more; (ii) the accuracy on tasks is improved: the agent can do things better; (iii) the speed is improved: the agent can do things faster.

In this paper, we discuss an approach to learning centered on the possibility of acquiring sets of rules from other agents, namely "learning by being told". We assume in fact that, whatever the formalism they are based upon, the agents that we are considering have a rule-based knowledge base. The acquired sets of rules can either define a reaction to a previously unknown event, or they can represent a plan to reach an objective. Indeed, learning from others is a fairly practical and economical way of increasing abilities, widely used by human beings: in fact, avoiding the cost of learning is an important benefit of imitation. An agent that learns and re-elaborates the learned knowledge may become itself an information producer, from which others can learn in turn.

However, agents should not blindly incorporate the new knowledge. Rather, they should be able to evaluate how useful the new knowledge is, and on this basis decide whether to keep or discard it. To this aim, we propose to associate to the acquired knowledge a specific objective and (possibly) a set of conditions, including a time limit. The purpose is that the new rules should help the agent reach that objective and fulfill the conditions within the given time limit. After a while, the agent will evaluate whether (or to what extent) this has been achieved. More sophisticated conditions/constraints that have to be verified can also be specified. If the evaluation is unsatisfactory, the new knowledge will be discarded or possibly deactivated for future re-trial in a modified context. There is a clear similarity between our approach and reinforcement learning, where here the action that is to be evaluated is the use of the new knowledge.

The approach is developed in the context of a general agent model proposed in [1], that provides high flexibility in how an agent is built and may evolve. In fact, we equip an agent with forms of understanding and awareness that are "situated" at different control layers. Beyond the basic control layer we introduce in fact a meta-control, where non-trivial supervising and tuning tasks can be performed, based on suitable control and meta-control information. It is at the meta-control level that we introduce the check for the evaluation of the acquired knowledge.

The present work was initiated by the experiments with the prototype implementation presented in [2], developed in DALI [3,4]. Later on, we have empowered the implementation with the temporal-logic-like operators that we introduced in [5,6], and we have performed some experiments in Ambient Intelligent applications [7]. In this paper, we present a further evolution of our approach, which as mentioned has been enriched with a meta-evaluation component, and generalized so as to make it usable in many rule-based agent-oriented approaches.

The paper is organized as follows. In Section 2 we further discuss motivations and potential usefulness of the proposed approach, and we outline the features of our proposal that we illustrate in more detail in Section 3. We introduce the semantics in Section 4. In Section 5 we propose a case-study, formalized in the language of our implementation: namely, we define an artificial fish able to learn from its shoal what to do in certain situations. Finally, we discuss related work and conclude in Section 6. In the Appendix we present the DALI language, as in fact our approach has been implemented in DALI and is under some respects inspired to features of the DALI language and architecture.

2 Motivations and Overall Framework

Learning may allow agents to survive and reach their goals in environments where a static knowledge is insufficient. The environmental context can change, cooperative or competitive agents can appear or disappear, ask for information, require resources, propose unknown goals and actions. Then, agents may try to improve their potential by interacting with other entities so as to perform unknown or difficult tasks.

One of the key features of MAS is the ability of "sub-contracting" computations to agents that may possess the ability to perform them. More generally, agents can try to achieve a goal by means of cooperative distributed problem-solving. However, on the one hand not all tasks can be delegated, and on the other agents may need or may want to acquire new abilities to cope with unknown situations. In our view, an improvement in the effectiveness of MAS may consist in introducing a key feature of human societies, i.e., cultural transmission of abilities. Without this possibility, agents are limited under two important respects:

- they are unable to expand the set of perceptions they can recognize, elaborate on and react to;
- they are unable to expand their range of expertise.

In fact, a well-known and particularly difficult problem in AI is the so-called "brittleness" problem: automated systems tend to "break" when confronted with even slight deviations from the situations specifically anticipated by their designers. Indeed, the flexibility and thus the "intelligence" of agents may increase if they become able not only to refine but also to enlarge their own capabilities. The need of acquiring new knowledge can be recognized by an agent at least in relation to the following situations:

1. There is an objective that the agent has been unable to reach: it has been unable to relate a plan (in the KGP perspective [8]) or intention (in a BDI perspective [9]) to that objective (or desire) and it has to acquire new knowledge (beliefs)[1]. As a particular case, there is a situation the agent is unable to cope with; for instance, there is an exogenous event that the agent does not recognize.
   Use case. An agent located in a network router for security management has to recognize and respond to attacks, and perform repairs. It is impossible to delegate this tasks, it is instead possible in some cases to learn suitable behaviors from trusted agents. For instance, the agent may perceive an unknown external event that presumably corresponds to an unknown attack: it can try to learn how to respond. Also, the agent can recognize an attack, and devise a plan for response and repair, but some step of this plan fails: the agent can try to learn alternative ways to reach the objective.
2. There is some kind of computation that the agent is unable to perform.
   Use case. An agent which acts as a mediator in a data integration context is responsible for reformulating, at runtime, queries defined on a (virtual) mediated schema into the local(actual) schemata of the underlying data sources. Most mediator systems use wrappers to handle the task of data access, data retrieval, and data translation: a wrapper module is able to access specific data sources, extract selected data, and translate source data formats into a common data model designed for the integration system. If a new data source is added to the system, a suitable wrapper may be missing. Delegating the wrapping is unpractical and would in most cases unacceptably spoil efficiency. The agent can try to acquire the wrapper from another and thus "learn" how to cope with the new data format.

Assuming that the agent establishes that it cannot resort to cooperation to get its task performed, it can still resort to cooperation in order to try to acquire the necessary piece of knowledge from another agent. The problems involved in this issue are at least the following: how to ask for what the agent needs; how to evaluate the actual usefulness of the new knowledge; and, how this kind of acquisition can be semantically justified in a logical agent.

[1] By some abuse of notation, we use the words 'knowledge' and 'belief' as synonyms when referring to agents. In fact, we implicitly consider autonomous agents that are in general not able to resort to 'knowledge' as such but only have their possibly subjective 'beliefs'.

In this paper, we make the simplifying assumption that there are ways for asking other agents and obtaining the potentially needed new knowledge. In our view it should not in principle be required that the involved agents be based on the same inference mechanism. It is reasonable however to assume that they are somehow "compatible": in particular, we assume that all the involved agents are rule-based and are able to incorporate prolog-like sets of rules.

In order to show that indeed we do not neglect this issue, we briefly summarize the learning rules process that we have defined and experimented in the prototype implementation of [2]. This solution is based on the introduction of a *mediator agent* that we called $yellow\_rules\_agent$, keeping track of the agents specialization and reliability, and of which pieces of knowledge they are willing to share with others. When an entity needs to learn something, it asks the $yellow\_rules\_agent$ for the names of agents having a certain specialization and being more reliable, that have declared to possess the needed piece of knowledge. In particular, in our implementation the requester provides the $yellow\_rules\_agent$ with a set of keywords that the latter will try to match with the current contents of its directory.

Once obtained this information, the agent may acquire the desired knowledge by some of them. If the agent will finally decide to incorporate the learned rules in its program because they work correctly, it will also send to $yellow\_rules\_agent$ a message indicating satisfaction. This will result in an increment of the reliability of the agent that has provided the rules. A negative experience will imply an unfavorable dispatch. In the present implementation agents return a numeric value indicating the "level of trust". In updating the level of trust, our $yellow\_rules\_agent$ adopts a model that updates trust only when the information is sufficient, i.e., after a certain number of reports which are in accordance, sent by reliable agents (we have treated this topic at some length in [10]).

However, as described in the Appendix the DALI language provides a communication layer where meta-rules can be defined so as to filter incoming and out-coming communications. These meta-rules are defined independently of the main agent program, so that the communication behavior of an agent can be "tuned" in an elaboration-tolerant way. This allows one to equip an agent with specific meta-rules, according to the strategy of trust management and knowledge retrieval that one wants to adopt. Therefore, our implementation is easily customizable to accommodate any strategy.

In summary, when an agent needs some knowledge it will first obtain the names of one or more agents that it is possible to ask for missing rules. Then it will contact (some of) them according to both the society's and the personal reliability evaluation of these agents. Finally, it will get the required item in a standard format (see e.g. the case-study in Section 5) so as to be able to use it. The exchanged piece of knowledge should include all the *relevant rules* in the sense of [11], i.e., all the rules which are required (directly or indirectly) for actually using that knowledge. Also, acquired rules should be submitted to a suitable renaming process so as not to overlap or interfere with the original knowledge base of the receiver agent.

At this stage, the receiver agent has to face two problems:

(a) establish whether the new knowledge is consistent, or at least compatible, with its knowledge base, and also self-consistent given the facts and the knowledge base.

This is a topic which has long been studied in belief revision [12], and that we do not discuss here.

(b) establish whether the new knowledge is actually useful to the purposes for which it has been acquired. If so, it can possibly be definitely asserted in the knowledge base. Otherwise, it can possibly be discarded.

Thus, agents should be able to evaluate how useful the new knowledge is. Similarly to reinforcement learning, techniques must be identified so as to make this evaluation feasible with reasonable efficiency. The discussion of heuristics for the evaluation is outside the scope of this paper, where we intend to outline the general setting, that can however accommodate several possibilities in a modular fashion. In fact, as discussed in the next section, the evaluation will be performed by means of a set of meta-rules that can be modularly defined independently of the "main" agent program. However, in general in our framework usefulness will not be evaluated by a simulation (which would be time-costly and would possibly worsen the problem of brittleness), rather it will be evaluated based on practical usage. This makes the topic of trustworthiness particularly relevant, as knowledge obtained by reliable sources is assumed to be maybe "not so good" but however not harmful for the receiving agent. Simple techniques to cope with this problem that we have already to some extent experimented in our prototype implementation are the following.

1. The new knowledge has been acquired in order to cope with an unknown event so as to fulfill some conditions: the agent can confirm/discharge the new knowledge according to the conditions being satisfied or not.
2. The new knowledge has been acquired in order to reach an objective: the agent can confirm/discharge the new knowledge according to its reaching/not reaching the objective. This evaluation can be related to additional parameters, like e.g. time, amount of resources needed, quality of results.
3. The new knowledge has been acquired for performing a computation: results which are not "sufficiently good" (given some sort of evaluation) lead to the elimination of the related piece of knowledge. The agent can possibly acquire the same type of knowledge from several sources, and compare/combine the results.

When some new knowledge is received, it will be used by the agent, one or more times, to the aim for which it has been acquired. After these trials, the receiver agent is able to assess the usefulness of the new knowledge, and decide whether it can be confirmed (i.e., permanently added to the knowledge base) or discarded. Possibly, rather than totally discarded the new knowledge can be de-activated for future possible re-trial or usage in a modified context.

The main point of our approach and the novel aspect proposed in this paper is that a meta-level device will be responsible of checking the usefulness of the new knowledge. Meta-level axioms will state on which basis the knowledge has to be evaluated. In fact, it is widely recognized (see, e.g., [13]) that coping with unexpected situations involves, for an agent, meta-level monitoring, reasoning about, and, when necessary, altering its own behavior. The agent meta-control, based on a meta-history, will then determine assimilation or discarding or deactivation of the new knowledge, by performing the checks for deciding whether to keep it.

Though our experiments have been performed in DALI, we do not intend to commit to a specific agent formalism or language. In fact, the semantic that we propose in Section 4 is general enough to accommodate many agent-oriented rule-based approaches.

3 The approach

In this Section we illustrate the specific features of the proposed approach. Let A be an agent defined according to some logic-based agent model $\mathcal{M}$. Assume that A has reached some stage of its operation where it recognizes the need of acquiring new knowledge from the outside in order to cope with a situation that cannot be managed by means of the knowledge which is presently available. A learning step is thus in order.

We introduce the possibility for the agent to learn reactive rules and plans. Once acquired, the new knowledge is stored in two forms.

– As plain knowledge added to the set of beliefs, so that the agent is able to use it.
– As meta-information, that allows the agent to "trace" the new knowledge, in the sense of recording what has been acquired, when and with which expectations. The meta-information allows the agent to perform meta-reasoning on these aspects. If the agent should conclude that the new rules must be removed because the expectations have not been met, the meta-information will be used to locate the rules in the beliefs and remove them.

The syntax that we adopt both in this Section and in the case-study of Section 5 is often reminiscent of logic programming: variables in upper case, constants and predicates in lower case, connective $\leftarrow$ (or :- like in many practical systems) between the head (conclusion) of a rule, and its body (conditions). Syntax is also reminiscent of the DALI language, where some new connectives are introduced (e.g., :> is a new connective that defines a reactive rule).

However, this syntax (which is the syntax of our implementation) is in general terms by no means mandatory, as it is basically aimed at illustrating on the one hand the conceptual elements of the approach and on the other hand how it can be put at work. A suitable variant of the syntax can be developed when applying the approach to some other practical setting.

We may assume for instance that the meta-information associated to a set of rules that an agent learns from the outside for coping with a previously unknown event has the following form:

react(R, event(E), rules(R1,...,Rn), cond(pos(P), neg(N)), time(T))

where R is an identifier for this set of rules (a constant); E specifies the event to be coped with; $R1, \ldots, Rn$ are the acquired reactive rules plus their required auxiliary rules; *cond* specifies in the *pos* part the positive condition(s) that have to be fulfilled after reaction ensues, and in the *neg* part the negative conditions; time specifies the time threshold allowed for condition fulfillment. For example:

react(r1,event(rains), rules((head(rains),body(open_umbrella))),
cond(pos(true),neg(wet)), time())

means that $r1$ is a rule acquired for coping with external event of rain, and that the condition to be fulfilled via the reaction is not getting wet. One rule is provided, in particular a reactive rule where the head is the event that has occurred, i.e., *rains* and the body specifies the action to be undertaken then, i.e., *open_umbrella*.

Analogously, we assume that the meta-information associated to a set of rules that an agent learns from the outside and that represent a plan for coping with an objective that previously could not be reached has the following form:

plan(P, obj(O), steps(S1,...,Sn), cond(pos(P), neg(N)), time(T))

where P is an identifier for this plan (a constant); O specifies the objective to be reached via this plan; $S1, \ldots, Sn$ represent the steps of the plan plus the needed auxiliary rules; *cond* specifies in the *pos* part the positive condition(s) that have to be fulfilled while reaching for the objective, and in the *neg* part the negative conditions; time specifies the time threshold allowed for reaching the objective. Example:

plan(toAirport, objective(atAirport), steps(...),
cond(pos(moneySpent $\leq$ *200), neg(lostPlane)), time(18:30))*

meaning that plan *toAirport* is aimed at getting at the airport while spending less that an amount 200 and avoiding to lose the plane.

Supervising activities will in general rely upon a *meta-history* generated during the agent's operation, that integrates in time the existing meta-control information. The meta-history should contain at least a list of:

- which goals have been set and at which time;
- which goals resulted in being successful/failed/timed-out and at which time;
- which incoming external events were known to the agent (and thus have been reacted to) and which ones were unknown instead.

The basic supervising activity can be based upon a mechanism similar to that of the *internal events* of the DALI logic programming agent-oriented language [3,4]. I.e., expectations related to each piece of new knowledge are (automatically) checked from time to time, and actions are undertaken on awareness of their violation.

Various properties that should be respected by the agent behavior can be expressed over the meta-history, also in terms of (adapted versions of) temporal statements, such as those introduced in [5,6]. When goal g is set, a record *goal_set*$(g) : t_1, t_2$ is added to the meta-history, meaning that the agent has decided at time t_1 to pursue this goal, that should be achieved by time t_2.

Whenever a goal is either reached, or failed or timed-out, a record of the form *successful*$(g) : t$ or *failed*$(g) : t$ or *timed_out*$(g) : t$ is added to the meta-history, where t is the time where the meta-conclusion about the outcome has been reached. Assume that time-stamps can be omitted if not needed.

Whenever an external event e that reaches the agent is recognized, a record of the form *known*$(e) : t$ is added to the meta-history, where t is the time when the event occurred. If instead the event is not recognized, a record of the form *unknown*$(e) : t$ is instead added.

4 Semantics of Learning by Rule Exchange

We adopt here the general agent model of [1], which does not stick to any specific approach for defining logical agents. Rather, the specific agent model $\mathcal{M}$ that one intends to embrace is an "input parameter" of the overall framework. As we will see however, it is defined in terms of components that, together, constitute and agent program, and is thus particularly suitable to represent rule-based agents.

An *agent* in this framework is characterized by an agent program $\mathcal{P}$ (defined in terms of the specific instance agent model $\mathcal{M}$) and a suitable underlying operational mechanism $\mathcal{U}$ that can be understood as an implementation which is able to run the agent program. $\mathcal{P}$ encompasses an explicit control component $\mathcal{C}$, operating on suitable control information $\mathcal{CI}$.

To the aims of the approach that we are introducing in this paper, below we augment the framework of [1] by introducing a meta-control component $\mathcal{MC}$, operating on suitable control information $\mathcal{MCI}$. Correspondingly, the underlying operational mechanism is enriched by a meta-control mechanism $\mathcal{H}$. Formally:

Definition 1. *Let $\mathcal{M}$ be an agent model. An agent program or simply "agent" $\mathcal{P}$ is a tuple $\langle \mathcal{B}, \mathcal{DI}, \mathcal{SC}, \mathcal{BM}, \mathcal{CS}, \mathcal{A}, \mathcal{C}, \mathcal{CI}, \mathcal{MC}, \mathcal{MCI} \rangle$ of software components where: $\mathcal{B}$ is the set of the agent's beliefs; $\mathcal{DI}$ the set of desires and intentions; $\mathcal{SC}$ is the sensing and communication component; $\mathcal{BM}$ is the belief management; $\mathcal{CS}$ a set of constraints; $\mathcal{A}$ is the set of actions that the agent has devised to perform; $\mathcal{C}$ is the object-level control component and $\mathcal{CI}$ the control information; $\mathcal{MC}$ is the meta-control component and $\mathcal{MCI}$ the meta-control information. Each component of the tuple is defined (or omitted) according to $\mathcal{M}$.*

The operational behavior of the agent will result from the control and meta-control components $\mathcal{C}$ and $\mathcal{MC}$ given the control and meta-control information $\mathcal{CI}$ and $\mathcal{MCI}$. In general, this information will be partly specified in advance and partly updated/generated later. The agent actual functioning in the environment where its is situated relies on underlying control and meta-control mechanisms $\mathcal{U}$ and $\mathcal{H}$ that implement the practical counterpart of the agent model.

Definition 2. *Let $\mathcal{M}$ be an agent model and $\mathcal{P}$ an agent program. Let the initial agent $A_0 = \mathcal{P}$ Let $\mathcal{E} = \{E_0, \ldots, E_n\}$ be a sequence of sets of events. The* underlying control mechanism *$\mathcal{U}$ of $\mathcal{M}$ is a transformation function that transforms (E_0, A_0) step by step into a sequence of agents $A_1, \ldots, A_n$. This transformation exploits the events E_i and the components $\mathcal{C}_i$ and $\mathcal{CI}_i$ of every agent A_i ($i \geq 0$):*

$$(E_i, A_i) \xrightarrow{\mathcal{U}(\mathcal{C}_i, \mathcal{CI}_i)} A_{i+1}$$

The meta-control acts by means of single steps, similarly to the control. Then, given as before an agent program $\mathcal{P}$, and an initial agent $A_0 = \mathcal{P}$:

Definition 3. *Let $\mathcal{E} = \{E_0, \ldots, E_n\}$ be a sequence of sets of events. The* underlying meta-control mechanism *$\mathcal{H}$ of $\mathcal{M}$ is a transformation function that transform (E_0, A_0)*

into a sequence of agents $A_1, \ldots, A_n$. This transformation exploits the events E_i and the components $\mathcal{MC}_i$ and $\mathcal{MCI}_i$ of every agent A_i ($i \geq 0$):

$$(E_i, A_i) \xrightarrow{\mathcal{H}(\mathcal{MC}_i, \mathcal{MCI}_i)} A_{i+1}$$

Based on suitable (meta-)control information, the meta-control can be exploited either in a domain-dependent or in a domain-independent fashion for supervising, checking, tuning many aspects.

Definition 4. *Let $\mathcal{M}$ be an agent model and $\mathcal{P}$ an extended agent program. Given a sequence of sets of events $\mathcal{E} = \{E_0, \ldots, E_n\}$, the* operational behavior *of $\mathcal{P}$ is defined as a sequence of transformation steps interleaving control and meta-control.*

We thus assume to perform some steps of meta-control after a number of steps of control. We do not specify here how many these steps are: they may be specified either in advance (built-in in $\mathcal{U}^{\mathcal{M}}$ and $\mathcal{H}^{M}$) or in the control information.

Our approach to rule exchange fits in this semantic framework: in fact, the history and the meta-history will be included into the control and meta-control information $\mathcal{CI}$ and $\mathcal{MCI}$. Among the actions devised by an agent at each step there may be a request for new rules to other agents. An incoming event can be the arrival of such new rules that will be managed, as exemplified in Section 5, by the meta-control $\mathcal{MC}$ and thus made available to the control component $\mathcal{C}$.

For the declarative semantics, we refer to the general setting introduced in [14]. where changes, either external (e.g., agent's reception of exogenous events) or internal (e.g., courses of actions undertaken based on internal conditions) are considered as producing a corresponding change in the agent program, which is a logical theory, and in its semantics (however defined). For such a change to be represented, we understand this change as the application of a program-transformation function. Also belief revision can be seen as a step of program transformation that in this case results in the updated theory.

In order to cope with adding and deleting the new knowledge we rely on the approach of EVOLP [3], that allows (sets of) rules to be conditionally added or deleted from a program. The EVOLP approach can be smoothly merged into our semantics: some of the evolution steps determined by the meta-control will be (a series of) EVOLP steps that imply requiring, adding or dropping some knowledge pieces.

5 Case Study: an Artificial Fish

The case-study that we consider is in the realm of adaptive controllers, where an adaptive controller can change its behavior in response to changes in the dynamics of the process and the disturbances [15]. In particular, we consider hybrid control systems, i.e., systems whose behavior is defined by processes of diverse characteristics. In our setting, such a controller is modeled ad depicted in Fig 1.

Its architecture consists of two loops. The inner loop is an ordinary feedback loop composed of the process and the controller. The behavior of the controller is adjusted by the outer loop which consists of a supervisory controller, that coincides with the meta-control component. In this context, we assume that the rules included in the controller

Fig. 1. Supervisory control system architecture

are not completely available from the beginning, but instead are learned when needed and then evaluated by the supervisor.

We consider as a scenario a virtual marine world inhabited by a variety of fish. They autonomously explore their dynamic world in search for food. Hungry predator fish stalk smaller fish who scatter in terror. For simplicity, the behavior of a fish is reduced to eating food and escaping, and is determined by the motivation of it being satiated and safe. In [16], McCarthy considers that "A fish cannot take instruction from a more experienced fish in how to swim better" as an example of a handicap that prevents fish from improving their behavior. Next, we show how our artificial fish can overcome this obstacle by receiving instructions, say from members of its shoal. In the reality, it will most presumably just imitate its mates. In our setting, this behavior is realized by asking for rules that will help it cope with an unknown situation.

Each fish is described by variables with values in the range $[0\ 1]$ with higher values indicating a stronger desire to eat or to avoid predators. In the formalization, we let t denote the clock time of the system.

- *hungry* : it expresses how hungry the fish is and it is approximated by

  $$\mathrm{hungry}(t) = \min\{\Delta T \times a, 1\}$$

  where ΔT denotes the time since the last meal and a indicates the appetite of the fish;
- *fear* : it quantifies the fear of the fish by taking into account the distance $d(t)$ of the fish to visible predators

  $$\mathrm{fear}(t) = \min\{D/d(t), 1\}$$

  where D indicates how coward the fish is.

The input vector to the controller is

$$\tilde{x}(t) = \begin{bmatrix} \mathit{hungry}(t) \\ \mathit{fear}(t) \end{bmatrix}$$

The fish behavior as well as its internal state (i.e., its beliefs) are modeled by means of an agent program

$$\mathcal{M} = \langle \mathcal{B}, \mathcal{C}, \mathcal{CI}, \mathcal{MC}, \mathcal{MCI} \rangle$$

where $\mathcal{B}$ is the fish's beliefs component (the remaining components are omitted as not necessary for this application). The controller and its supervisor are formalized via $\mathcal{C}$ and $\mathcal{CI}$, and $\mathcal{MC}$ and $\mathcal{MCI}$, respectively. Assume that at some state α the meta-control information component $\mathcal{MCI}_\alpha$ contains the rules

$$\begin{array}{ll} & \textit{prop1(E)} \leftarrow \textit{SOMETIMES not know(E)} \\ (r1) & \textit{prop1(E):>learn(new\_rule\_for(E))} \\ & \textit{know(hunger)} \end{array}$$

stating that any time there exists an unknown event, then a new rule to cope with that event must be learned. Suppose that at state α the fish knows that it has to search for food when it is hungry. This is formalized in $\mathcal{CI}_\alpha$ with the reactive rule (see Appendix):

$$\textit{hunger(X), X} \geq \textit{0.5, not food :> search(food)}$$

where X is the value of hungriness. Note that the stimuli of the fish (i.e., its input vector $\tilde{x}(t)$) are represented at the controller level via the notion of event. For example, the value v of the stimulus *hungry(t)* of the process is represented as *hunger(v)*.

Suppose that the fish perceives the stimulus of fear. Being this stimulus unknown, $\mathcal{MC}$ requires a new rule to handle the unknown event via the reactive rule (r1).

Assume that at a later state α_2, the meta-control receives in response to its request the rule:

$$\begin{array}{ll} (r2) & \textit{react(\#2, event(fear), rules(} \ulcorner \textit{fear(X), X} \geq \textit{0.5, nearby(predator) :> flee} \urcorner \textit{),} \\ & \textit{cond(pos(true), neg(nearby(predator))), time(10))} \end{array}$$

Here, $\ulcorner r \urcorner$ abbreviates the representation of a rule r and $\#2$ is a unique rule identifier. Rule r2 is a meta-rule, and is aimed at producing actual object-level rules to be employed by the fish. In particular, upon reception of r2 rules r4 and r5 are generated in a standard way and added to the fish belief base. r2 looks highly domain-specific, as it is supposed to be provided by some other fish of the shoal. However, its structure is general, and it can be seen as the instance of a meta-meta rule for encoding sets of rules to be shared with other agents. In this setting, we suppose that the supervisor also receives a related *evaluation rule* which is more detailed than r2 and declaratively expresses how to evaluate r2. In principle, different evaluation rules might be associated to the same learned meta-rule.

$$\begin{array}{ll} (r3) & \textit{eval(\#2, act(pos(nearby(predator)), neg(false)),} \\ & \textit{obj(pos(true), neg(nearby(predator))),} \\ & \textit{time(20), criticality(high), action(drop\_rule))} \end{array}$$

Abstracting away from domain-specific aspects, it can be seen that r3 is an instance of a meta-meta rule and in fact it states (in addition to what already expressed in r2): the activation conditions to start the evaluation (in this case, the simple fact that r2 has been received); the objectives that need to be achieved as well as the time interval to achieve them (in this case they add nothing to r2, but more conditions might be stated), the criticality level of the rule under evaluation; the action to be undertaken if the objectives are not fulfilled within the time constraints. From r3, rules r6-r9 below can be automatically generated.

Then, the extended agent program (representing the overall behavior of the fish) $A_{\alpha 2}$ evolves through a meta-control step as follows:

$$(E_{\alpha_2}, A_{\alpha_2}) \xrightarrow{\mathcal{H}(\mathcal{MC}_{\alpha_2}, \mathcal{MCI}_{\alpha_2})} A_{\alpha_2+1}$$

where

$E_{\alpha_2} = \{r2, r3\}$
$A_{\alpha_2} = \langle \mathcal{B}_{\alpha_2}, \mathcal{C}_{\alpha_2}, \mathcal{CI}_{\alpha_2}, \mathcal{MC}_{\alpha_2}, \mathcal{MCI}_{\alpha_2} \rangle$
$A_{\alpha_2+1} = \langle \mathcal{B}_{\alpha_2+1}, \mathcal{C}_{\alpha_2+1}, \mathcal{CI}_{\alpha_2+1}, \mathcal{MC}_{\alpha_2+1}, \mathcal{MCI}_{\alpha_2+1} \rangle$ (defined below)

The aim of this meta-control step is to incorporate the new learned rules into the extended agent program A_{α_2}. These new rules may be possibly de-activated later if they are considered not useful. When a rule is acquired it is assumed to be *active*. Every learned rule is exploited only if active. Whenever the supervisor takes the decision to drop a rule, it simply drops the assumption of the rule being active. This leaves the way open to a possible later re-activation of the rule.

As mentioned, the rules that are added at the object level so as to make the incoming rules r2 and r3 operative are rules r4-r12 (specified below in the same syntax that we have already employed in Section 3). Rule r4 states that the reactive rule related to r2 (and denoted by its identifier $\#2$) can be applied whenever: (i) it is active and (ii) the corresponding event (left-hand side) has occurred. In this case the reaction (right-hand side) will take place. Rule r5 states that $\#2$ is active. Rule r6 sets the objectives specified in the evaluation rule r3, while r7 asserts the related meta-history item. Rules r8 checks whether the objective has been achieved in time, while r9 asserts the related meta-history item.

Rules r10-r12 can be considered to be specific of this particular agent, and state what to do whenever the evaluation of knowledge acquired from outside is negative. In particular, rule r10 specifies that an objective must never go timed-out (which in this setting subsumes failure) and r11 states what to do if this requirement is violated: drop rule $\#2$. This will consist, as discussed above, in performing an $assert(not\ active(\#2))$ to de-activate the rule.

$(r4)$ *active(#2), fear(X), X $\geq$ 0.5, nearby(predator) :> flee*
$(r5)$ *active(#2)*

$(r6)$ *obj(#2, cond(pos(true), neg(nearby(predator))))* $\leftarrow$
nearby(predator), active(#2)
$(r7)$ *obj(#2,X), not obj_set(#2,_), current_time(T) :> assert(obj_set(#2,X):T,T+10)*

$(r8)$ *obj_achieved(#2):T* $\leftarrow$
obj_set(#2,cond(pos(P),neg(N))):T1,T2,
P, not N, current_time(T), T $\leq$ T2
$(r9)$ *obj_achieved(#2):T :> assert(obj_achieved(#2):T)*

$(r10)$ *prop3* $\leftarrow$ *NEVER obj_set(#2,_), timed_out(#2)*
$(r11)$ *not prop3 :> drop(#2)*
$(r12)$ *timed_out(#2)* $\leftarrow$
not obj_achieved(#2), obj_set(#2,_):T1,T2
current_time(T), T $>$ *T2*

The extended agent program A_{α_2+1} is therefore defined as follows (where in EVOLP notation $\circ$ denotes rule assertion):

$$\begin{array}{l}
\mathcal{CI}_{\alpha_2+1} = \mathcal{CI}_{\alpha_2} \circ \{r4, r5\} \\
\mathcal{MCI}_{\alpha_2+1} = \mathcal{MCI}_{\alpha_2} \circ \{r5 - r12\} \\
\text{and } \mathcal{MC}_{\alpha_2+1} = \mathcal{MC}_{\alpha_2} \\
\mathcal{B}_{\alpha_2+1} = \mathcal{B}_{\alpha_2} \circ \{\} \\
\mathcal{C}_{\alpha_2+1} = \mathcal{C}_{\alpha_2} \circ \{\} \\
\mathcal{MC}_{\alpha_2+1} = \mathcal{MC}_{\alpha_2} \circ \{\}
\end{array}$$

That is, A_{α_2+1} is obtained by updating (wrt. the EVOLP semantics) the components of A_{α_2} with the specified sets of rules.

In this kind of setting, preferences/priorities among events are particularly important (in DALI, such priorities can be provided in the initial control information associated to an agent program). In fact, in our example fear must be given higher priority than hunger. Otherwise, paradoxical behavior may arise: suppose that in a later state α_3, the controller and the supervisor perceive high values for both the fear and the hunger stimuli, and the event that a predator is nearby, that is, $E_{\alpha_3} = \{$*nearby(predator), fear(0.7), hunger(0.6)*$\}$. In such a situation, the controller has two alternative choices: either search for food or flee. Suppose that, without priorities having been stated, it selects the first alternative. Then, it is easy to see that if the time interval of 10 passes by, *time_out(#2)* holds in $\mathcal{MCI}_{\alpha_3+i}$, for some i. Being the objective set for $\#2$, *prop3* holds by rule r10. This will trigger the reactive rule r11 thus resulting in the removal (more precisely, in the de-activation) of rule r4. That is, $\mathcal{CI}_{\alpha_3+i+1}$ would be $\mathcal{CI}_{\alpha_3+i} \circ \{$*assert(not active(#2))*$\}$.

6 Related Work and Concluding Remarks

We have proposed an approach that allows logical agents to adopt a form of learning which consists in improving each agent's skills by acquiring new knowledge from other agents. We believe that this kind of technique can be often useful, and in some application contexts it can even be a key feature.

The problem that we have tackled here is specific to the particular realm of agents, that are able to acquire knowledge from other agents, i.e., from the "society" to which they belong. This is not in contrast with an agent adopting direct ("deep") learning techniques, rather it is complementary. In fact, as deep learning is time-consuming and costly, each agent in a society may apply a combination of deep learning and imitation.

We have illustrated a specific instance of the approach, that we have implemented in DALI, and we have discussed a case-study. The implementation, though prototypical and specific for DALI agents, constitutes a proof-of-concept for the effectiveness of the approach. However, in the overall framework that we have depicted we did not stick to a specific formalism or language for logical agents. In fact, the approach and its semantics are general enough to allow for a wide applicability.

Our approach brings some similarity with the approach of [17,18]. There, a BDI agent not possessing a plan to manage an event, is able to ask agents from a certain set S for such a plan. Symmetrically, an agent can define each of its plans as private, public, or sharable with a set of trusted agents. This copes with the problem of where to find the needed knowledge. Their approaches has been implemented and applied for instance to the scenario of service-oriented computing. Our approach adds the aspect of meta-reasoning for evaluating, activating and de-activating the new knowledge, where this evaluation may in principle affect the level of trust of source agent.

The works in [19,20] are aimed, again in a BDI context, at filtering new percepts according to their expected relevance to the current agent's ongoing desires and intentions. This may also help an agent to understand when to reconsider her deliberations. The latter proposal adopts meta-reasoning techniques for doing so. The principles and methods outlined in these works might be suitably integrated in our approach so as to evaluate how relevant the acquired new knowledge can be to the current context: in our experiments we considered quite rough methods for knowledge evaluation, while the mentioned approaches propose more involved evaluation techniques. As however our approach is modular w.r.t. this aspect, we can in future work implement such techniques in the communication layer of our architecture. In fact, [20] explicitly advocates this kind of meta-reasoning to be performed in a preprocessing module on incoming information. Future work will certainly be concerned with more involved techniques for knowledge retrieval and evaluation. To this aim, we intend to design meta-meta levels for controlling knowledge exchange.

Finally, we intend to fully implement an instance of the proposed framework, accommodating not only DALI agents but also other kinds of agents and architectures.

References

1. Costantini, S., Tocchio, A., Toni, F., Tsintza, P.: A multi-layered general agent model. In: AI*IA 2007: Artificial Intelligence and Human-Oriented Computing, 10th Congress of the Italian Association for Artificial Intelligence. LNCS 4733, Springer-Verlag, Berlin (2007)
2. Costantini, S., Tocchio, A.: Learning by knowledge exchange in logical agents. In: From Objects to Agents: Intelligent Systems and Pervasive Computing, Proc. of WOA'05. (2005) ISBN 88-371-1590-3.
3. Costantini, S., Tocchio, A.: A logic programming language for multi-agent systems. In: Logics in Artificial Intelligence, Proc. of the 8th Europ. Conf.,JELIA 2002. LNAI 2424, Springer-Verlag, Berlin (2002)
4. Costantini, S., Tocchio, A.: The DALI logic programming agent-oriented language. In: Logics in Artificial Intelligence, Proc. of the 9th European Conference, Jelia 2004. LNAI 3229, Springer-Verlag, Berlin (2004)
5. Costantini, S., Dell'Acqua, P., Pereira, L.M.: A multi-layer framework for evolving and learning agents. In: Proc. of the AAAI-08 Workshop on Metareasoning: Thinking about Thinking, Stanford University, AAAI Press (2008)

6. Costantini, S., Dell'Acqua, P., Pereira, L.M., Tsintza, P.: Runtime verification of agent properties. In: Proc. of the Int. Conf. on Applications of Declarative Programming and Knowledge Management (INAP09). (2009)
7. Costantini, S., Dell'Acqua, P., Pereira, L.M., Toni, F.: Learning and evolving agents in user monitoring and training. In: Proc. of the AICA Italian Conference, L'Aquila, Italy. (2010)
8. Bracciali, A., Demetriou, N., Endriss, U., Kakas, A., Lu, W., Mancarella, P., Sadri, F., Stathis, K., Terreni, G., Toni, F.: The KGP model of agency: Computational model and prototype implementation. In: Global Computing: IST/FET International Workshop, Revised Selected Papers. Volume 3267 of LNAI. Springer-Verlag, Berlin (2005) 340–367
9. Rao, A.S., Georgeff, M.: Modeling rational agents within a bdi-architecture. In: Proc. of the Second Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'91), Morgan Kaufmann (1991) 473–484
10. Costantini, S., Tocchio, A., Verticchio, A.: Communication and trust in the DALI logic programming agent-oriented language. Intelligenza Artificiale, J. of the Italian Association of Artificial Intelligence **2**(1) (2005) (in English).
11. Dix, J.: A classification theory of semantics of normal logic programs: I. strong properties. Fundamenta Informaticae **22**(3) (1995)
12. Antoniou, G.: Nonmonotonic Reasoning. ISBN 0-262-01157-3. The MIT Press, Cambridge, Massachusetts (1997) with contributions by M.-A. Williams.
13. Anderson, M.L., Perlis, D.R.: Logic, self-awareness and self-improvement: The metacognitive loop and the problem of brittleness. Journal of Logic and Computation **15**(1) (2005)
14. Costantini, S., Tocchio, A.: About declarative semantics of logic-based agent languages. In Baldoni, M., Torroni, P., eds.: Declarative Agent Languages and Technologies. LNAI 3229. Springer-Verlag, Berlin (2006) Post-Proc. of DALT 2005.
15. Åstrom, K.J., Wittenmark, B.: Computer-Controlled Systems.Theory and Design. Prentice Hall Internal Inc. (1990)
16. McCarthy, J.: Making robots conscious of their mental states. In: Machine Intelligence 15. (1995) 3–17
17. Ancona, D., Mascardi, V., Hübner, J.F., Bordini, R.H.: Coo-agentspeak: Cooperation in AgentSpeak through plan exchange. In: 3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004), 19-23 August 2004, New York, NY, USA, IEEE Computer Society (2004) 696–705
18. Bozzo, L., Mascardi, V., Ancona, D., Busetta, P.: Coows: Adaptive BDI agents meet service-oriented computing. In: EUMAS 2005 - Proceedings of the Third European Workshop on Multi-Agent Systems, Brussels, Belgium, December 7-8, 2005, Koninklijke Vlaamse Academie van Belie voor Wetenschappen en Kunsten (2005) 473
19. Lorini, E., Piunti, M.: Introducing relevance awareness in bdi agents. In: Proc. of the 7th international conference on Programming multi-agent systems (ProMAS 2009). Volume 5919 of LNCS., Springer (2010) 219–236
20. Koster, A., abd F. Dignum, F.K., Sonenberg, L.: Augmenting bdi with relevance: Supporting agent-based, pervasive applications. In: Proc. of Pervasive Mobile Interaction Device. (PERMID 2008). (2008)
21. Costantini, S., D'Alessandro, S., Lanti, D., Tocchio, A.: DALI web site, download of the interpreter (2010) http://www.di.univaq.it/stefcost/Sito-Web-DALI/WEB-DALI/index.php, With the contribution of many undergraduate and graduate students of Computer Science, L'Aquila. For beta-test versions of the interpreter (latest advancements) please ask the authors.

A The DALI language and Architecture

DALI [3,4] is an Active Logic Programming language designed for executable specification of logical agents. The DALI interpreter is freely available [21]. A DALI agent is a logic program that contains a particular kind of rules, reactive rules, aimed at interacting with an external environment. The environment is perceived in the form of external events, that can be exogenous events, observations, or messages by other agents. In response, a DALI agent can perform actions, send messages, adopt goals, etc. The reactive and proactive behavior of the DALI agent is triggered by several kinds of events: external events, internal, present and past events.

External events are syntactically indicated by the postfix *E*. When an event arrives to the agent from its "external world", the agent can perceive it and decide to react. The reaction is defined by a reactive rule which has in its head that external event (or, possibly, a conjunction of external events). The special token :>, used instead of :-, indicates that reactive rules performs forward reasoning. The agent remembers to have reacted by converting an external event into a *past event* (postfix *P*).

However, when an agent perceives an event from the "external world", it doesn't necessarily react to it immediately: it has the possibility of reasoning about the event, before (or instead of) triggering a reaction. In this situation, the event is called present event and is indicated by the postfix *N*.

In DALI, actions (indicated with postfix A) may have or not preconditions: in the former case, the actions are defined by actions rules, in the latter case they are just action atoms. An action rule is just a plain rule, but in order to emphasize that it is related to an action, we have introduced the new token :<, thus adopting the syntax $action\ :<\ preconditions$. Similarly to events, actions are recorded as past actions.

Internal events make a DALI agent agent proactive. An internal event is syntactically indicated by the postfix *I*, and its description is composed of two rules. The first one contains the conditions (knowledge, past events, procedures, etc.) that must be true so that the reaction (in the second rule) may happen. Thus, a DALI agent is able to react to its own conclusions. Internal events are automatically attempted with a default frequency customizable by means of directives in the initialization file.

The DALI communication architecture consists of four layers. The first layer implements the DALI/FIPA communication protocol and a filter on communication, i.e. a set of rules that decide whether or not to receive or send a message. The second layer includes a meta-reasoning user-customizable module that tries to understand message contents, possibly based on ontologies and/or on forms of commonsense reasoning. The third layer consists of the DALI interpreter. The fourth layer implements a filter for the out-coming and incoming messages.The DALI/FIPA protocol consists of the main FIPA primitives, plus few new primitives which are particular to DALI.